GPU Computing Using WebGL -Day 2

Wednesday, March 20, 2019 8:00 AM

Go to the following link

http://www.chaos.gatech.edu/ccis2019/sc1/

From Day 2, on the page, download the materials and the codes needed!

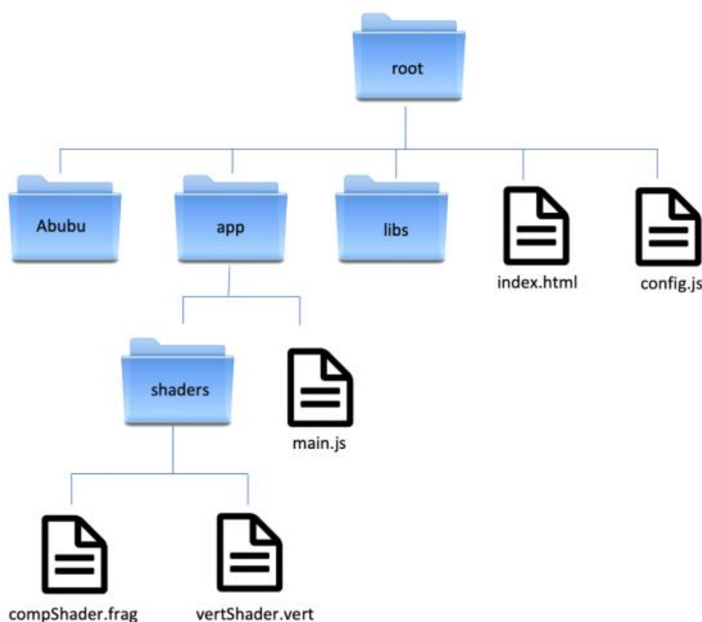
The objectives for today:

- Learning iteration basics in WebGL and calculating fractals
- Learning to render to textures (preparing you for solving PDEs)
- Using textures to plot computed data in Abubu.js;
- Introduction to basic user interactions using Abubu.js
- Study of complex iterative maps and fractals using WebGL

A good reference for today's work is this recent publication:
https://www.sciencedirect.com/science/article/pii/S0960077919300037

Quick reminder:

Our sample programs all have the following directory structure.



The source codes that we will be editing are mostly **main.js**, **vertShader.vert** and **compShader.frag**.

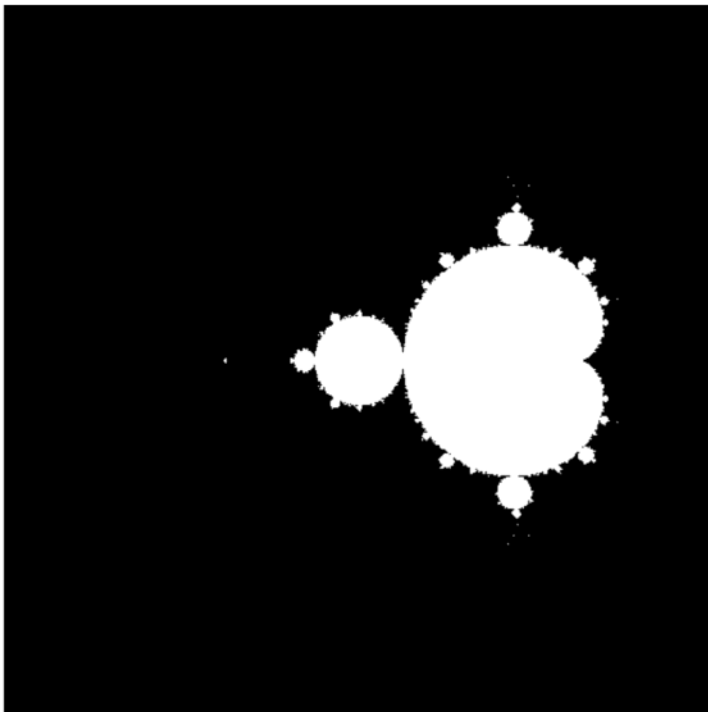
The Mandelbrot Set

We start from the content of the project 01-circle and build our Mandelbrot program.

So, let's start editing the fragment shader (compShader.frag):

```
#version 300 es
precision highp float ;
precision highp int ;

out vec4 outcolor ; // output color of the shader
in  vec2 pixPos ;   // input from vertex shader

void main() {
    vec2 z = pixPos*4. + vec2(-3,-2) ;
    vec2 c = z ;

    for (int i=0; i<200; i++){
        z = vec2(z.x*z.x - z.y*z.y,2.*z.x*z.y)+c ;
        if (length(z)>10.){
            outcolor = vec4(0,0,0,1.) ;
            return ;
        }
    }

    outcolor = vec4(1.) ;
    return ;
}
```

If we run the program by opening the index.html file in FireFox we should get the following result:

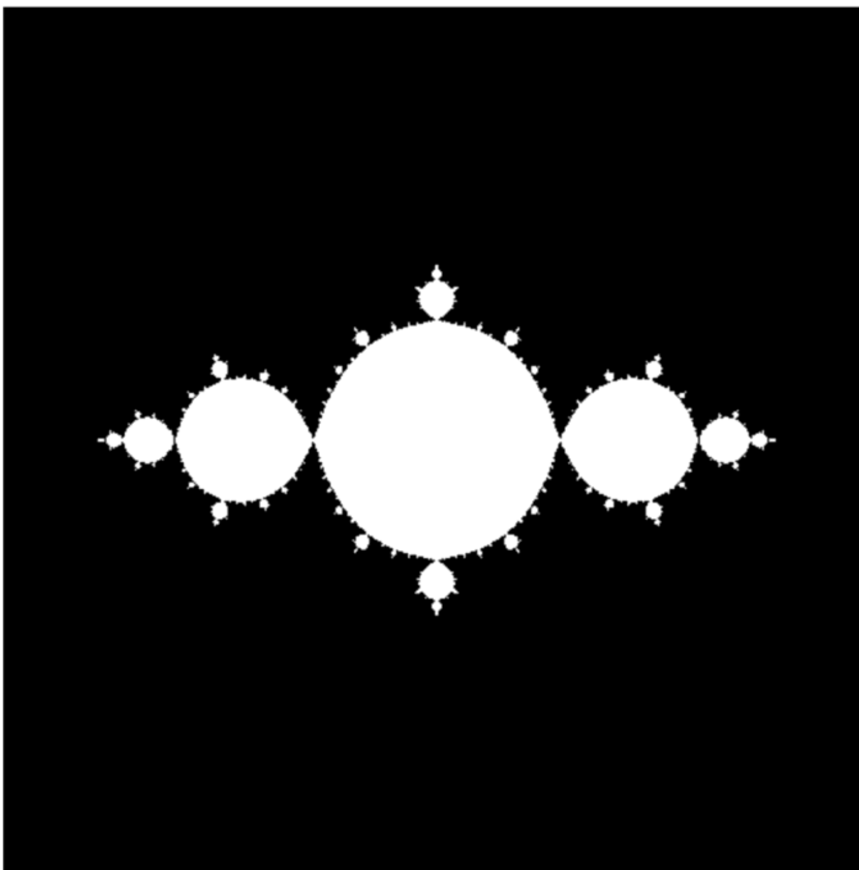


The Julia Set

Let's modify the Mandelbrot set fragment shader to get the Julia set.

```
#version 300 es
precision highp float ;
precision highp int ;

out vec4 outcolor ; // output color of the shader
in  vec2 pixPos ;   // input from vertex shader

#define  csqr(z)    vec2((z).x*(z).x - (z).y*(z).y,2.*(z).x*(z).y)

void main() {
    vec2 z = pixPos*4. + vec2(-2,-2) ;
    vec2 c = vec2(-.9,0.) ;

    for (int i=0; i<200; i++){
        z = csqr(z) + c ;
        if (length(z)>100.){
            outcolor = vec4(0,0,0,1.) ;
            return ;
        }
    }

    outcolor = vec4(1.) ;
    return ;
}
```

This fragment shader will create the following set:

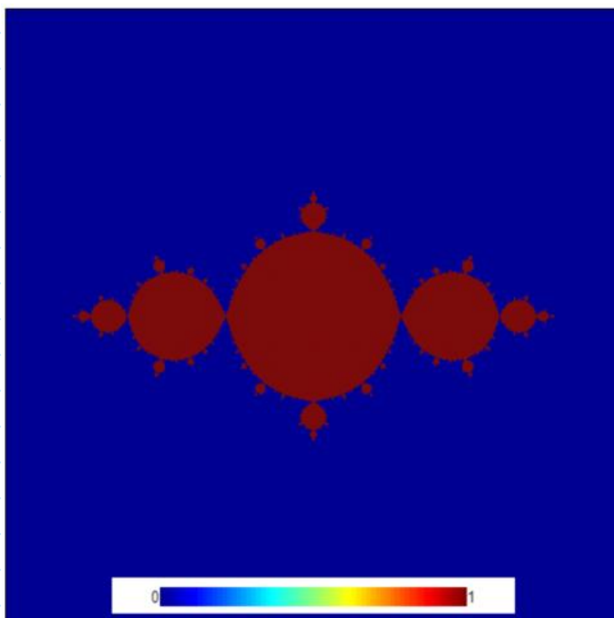


Rendering to textures and plotting using Abubu.js

Let's edit main.js to change the output of the solver to a texture/image/render target.

Moreover, let's using a plotting utility to display the results.

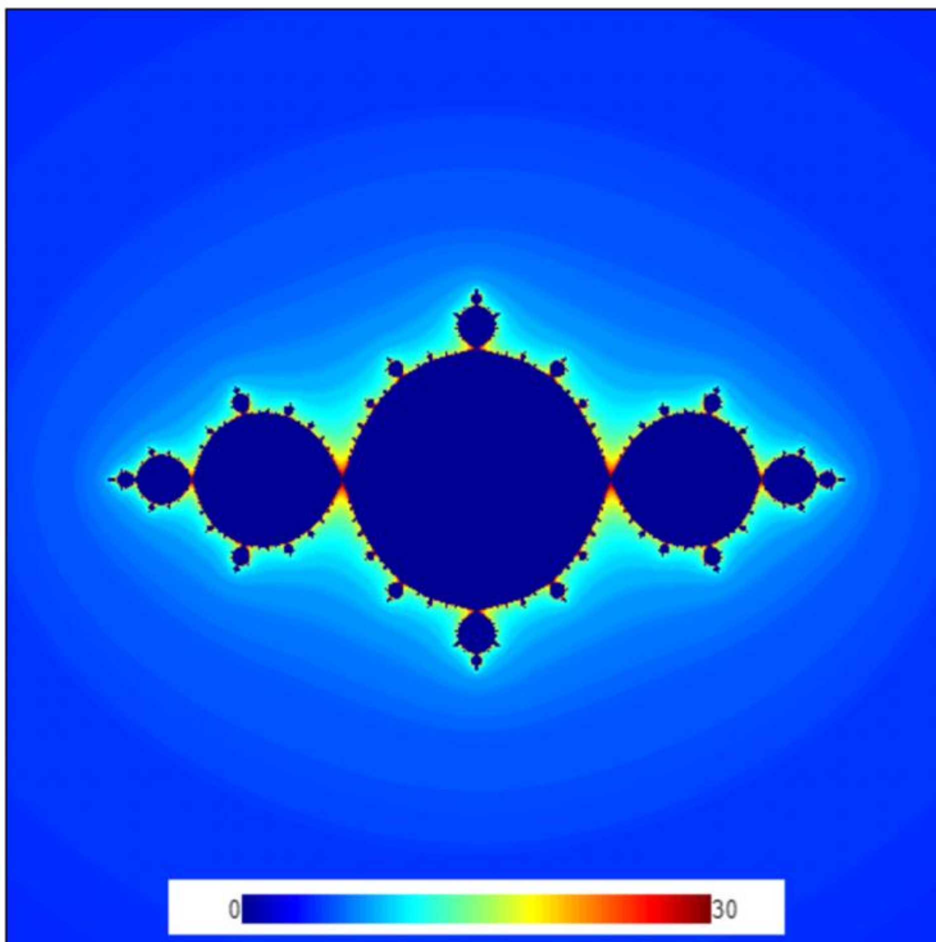
```
var txtr = new Abubu.Float32Texture(env.width,env.height) ;

/* Setup a solver */
var renderer = new Abubu.Solver( {
    fragmentShader   : compShader.value,
    vertexShader     : vertShader.value,
    renderTargets:{
        outcolor : { location : 0, target : txtr }
    }
} ) ;

var plt = new Abubu.Plot2D({
        target : txtr ,
        channel: 'r' ,
        minValue : 0 ,
        maxValue : 1 ,
        colorbar : true ,
        canvas   : canvas_1 ,
    } ) ;
plt.init() ;

function run(){
    renderer.render() ;
    plt.render() ;
}

run() ;
```



Let's add a coloring scheme

We can add a coloring scheme to our fragment shader so that we can get presentable fractals:

```
#version 300 es
precision highp float ;
precision highp int ;

out vec4 outcolor ; // output color of the shader
in  vec2 pixPos ;   // input from vertex shader

#define  csqr(z)    vec2((z).x*(z).x - (z).y*(z).y,2.*(z).x*(z).y)

void main() {
    vec2 z = pixPos*4. + vec2(-2,-2) ;
    vec2 c = vec2(-.9,0.) ;

    int iter = 0 ;
    for (int i=0; i<200; i++){
        iter = i ;
        z = csqr(z) + c ;
        if (length(z)>100.){
            break ;
        }
    }

    outcolor = vec4(float(iter) - log(log(length(z))/100.)) ;
    return ;
}
```



No webpage is complete without interaction

But, we need to get prepared. So, let's prepare our fragment shader with some uniform variables. The modified part of the shader is:

```
uniform  float rr, theta ;

#define  csqr(z)    vec2((z).x*(z).x - (z).y*(z).y,2.*(z).x*(z).y)

void main() {
    vec2 z = pixPos*4. + vec2(-2,-2) ;
    vec2 c = vec2(rr*cos(theta),rr*sin(theta)) ;
```

We need to set the uniforms in our solver. This happens in main.js (CPU-Side).

```
env.r = 0.6 ;
env.th = 0.1 ;

/* Setup a solver */
env.renderer = new Abubu.Solver( {
    fragmentShader  : compShader.value,
    vertexShader    : vertShader.value,
    uniforms : {
        rr : { type : 'f', value : .6 } ,
        theta : { type : 'f', value : 0.1 } ,
    },
    renderTargets:{
        outcolor : { location : 0, target : txtr }
    }
} ) ;

    env.plt = new Abubu.Plot2D({
            target : txtr ,
            channel: 'r' ,
            minValue : 0 ,
            maxValue : 30 ,
            colorbar : true ,
            canvas   : canvas_1 ,
        } ) ;
    env.plt.init() ;

    createGui() ;

    run() ;
}/*  End of loadWebGL  */
function run(){
    env.renderer.render() ;
    env.plt.render() ;
}

function createGui(){
    var gui = new Abubu.Gui() ;
    var pnl = gui.addPanel() ;
    pnl.add(env, 'r').step(0.01).onChange(function(){
        env.renderer.uniforms.rr.value = env.r ;
        run() ;
    } ) ;
    pnl.add(env, 'th').step(0.01).onChange(function(){
        env.renderer.uniforms.theta.value = env.th ;
        run() ;
    } )  ;
}
loadWebGL() ;
```